Partner Name 1: Travis Xie

Partner Name 2: Susan Yang

"Assignment 3 for CSE 415, Winter 2021, University of Washington"

Deterministic Simplified Backgammon Agent:

We met on zoom and discussed together how minimax works, and how to implement the minimax function, alpha-beta pruning as well as the static evaluation function. We checked with existing game platforms to make sure the static eval function accurately represents the goodness of states. We also did live share code on VS code and both contributed to parts of the code. Travis also worked to clean up the code and add comments.

How does the static evaluation function work?

We got the idea of pip counts from the online backgammon game. The first loop corresponds to all the checkers on the board. The farther away from a particular player's home base, the larger it will add to that player's pip value. For example, the white checker at position 0 will add 24 counts to the pip (the white home is from 18 - 23), and the white checker at position 18 will add 6 counts to the pip. So, at the beginning of the game, the total pip counts for both players are 169. We also want to take into account the possibility of a single checker on a certain position being hit by the opponent in the way. Since being hit could be greatly disadvantageous, so we take the number of possible opponent checkers that could hit our single checker and double it to add to our pip count.

Then, for each checker on the bar, it will add 25 to that particular player's pip count because it will take the checker 25 turns to arrive at 0 position. Lastly, for each checker already off the board, we subtract 25 from the pip count as a reward.

Consideration for alpha-beta pruning:

We updated the alpha or beta value for each bestValue we obtained from iteration (similar to the best first). Once the alpha is greater than the beta, the code will cut off the rest of the states.

Other comments on the implementation:

To get the resulting move from the minimax function, we used a global variable (a dictionary) to save the best move (as value) with the bestValue (as a key) at the end of the function call. This allows us to easily return the best move in `move()` function. Also, the dictionary will be reset every time when the move function is called to save space. In the `get_all_possible_moves_state()` function, if no move can be found, we will return the state passed in.

Stochastic Simplified Backgammon Agent:

We used a similar process as we did when writing the DSBG Agent. We looked at examples in class, and discussed how the `expectimax()` function should work and both tried different implementations and settled on what seemed right in our code.

Other comments on the implementation:

We used a similar implementation as the `minimax()` function for our `expectimax()` function. Again, to get the resulting move from `expectimax()`, we also used a global variable to store and retrieve the best move.

Partnership retrospective:

We initially didn't agree on how `expectimax()` should work, and couldn't convenience each other. But then we looked back at the slides and learned how it should work.

Lesson Travis learned as a result of working in this partnership:

Active communications can generate inspiration and improve working efficiency. Also, a great partnership can help refine understanding through discussion and explanation. In the working process, I gained a better understanding of some of the concepts.

Lesson Susan learned as a result of working in this partnership:

I should be more active in this partnership, this includes understanding the algorithm, studying the assignment, and thinking about the problem before the meetings. This will allow me to make more contributions, make our meetings more effective and our collaboration smoother. Next time I will be sure to do so before jumping into meetings.

Optional additional comments:

Compared to the DSBG agent, SSBG will take a longer time to run because it evaluates 36 possible dice roll combinations for each state in `move_state_list`. However, the SSBG agent will usually take fewer turns to win than the DSBG agent.